Evaluating the Vulnerability of Network Mechanisms to Sophisticated DDoS Attacks

Udi Ben-Porat[§]
Computer Science Dept.
Tel-Aviv University, Israel
Email: ehudben@post.tau.ac.il

Anat Bremler-Barr[§]
Computer Science Dept.
Interdisciplinary Center, Herzliya, Israel
Email: bremler@idc.ac.il

Hanoch Levy[*]
Computer Engineering & Networks Laboratory
ETH Zurich, Switzerland
Email: hanoch@tik.ee.ethz.ch

Abstract—**The design of computer and communication systems has been based, for decades, on the fundamental assumption that the objective of all users is to *improve their own performance*. In recent years we have experienced a wave of DDoS attacks threatening the welfare of the internet. These are launched by *malicious* users whose pure incentive is to *degrade the performance of other, innocent, users*. The traditional systems turn out to be quite vulnerable to these attacks.**

The objective of this work is to take a first step to close this *fundamental gap*, aiming at laying a foundation that can be used in future computer/network designs taking into account the malicious users. Our approach is based on proposing a metric that evaluates the vulnerability of a system. We then evaluate the commonly used data structure in network mechanisms, the hash data structure, using our vulnerability metric. We show that a Closed Hash is much more vulnerable than an Open Hash to DDoS attacks, even though the two systems are considered to be equivalent via traditional performance evaluation. We also apply the metric to queueing mechanisms common to computer and communications systems. Lastly we apply it to the practical case of a hash table whose requests are controlled by a queue, showing that even after the *attack has ended*, the regular users still suffer from performance degradation or even a total denial of service.

I. Introduction

In recent years the welfare of the Internet has been threatened by malicious attacks of distributed denial of service (DDoS). DDoS attackers consume the resources of the victim, a server or a network, causing a degradation in performance or even total failure of the victim.

The *basic DDoS attack* is a simple brute force flooding, where the attacker sends as much traffic as he/she can to consume the network resources, namely the bandwidth of the server's incoming link, without any knowledge of the system design. This can be achieved by sending a huge amount of traffic over a raw socket; or by using an army of Zombies[1] each mimicking the requests of a regular user.

More sophisticated attacks try to increase their effectiveness by aiming at hurting a weak point in the victim's system design, i.e. the attacker sends traffic consisting of complicated requests to the system. An example of such attacks is an attack against http servers by requesting pages that are rarely requested (forcing the server to search on the disk).

In this paper we concentrate on sophisticated attacks. We define *Sophisticated DDoS attacks* as attacks that increase their effectiveness by aiming at hurting a weak point in the victim's system design, i.e. the attacker sends traffic consisting of complicated requests to the system. This generalized definition covers all the different attack types described in recent papers [1], [2], [3], [4], [5], [6], [7].

Our first contribution in this paper (Section III) is a proposal of a new metric that evaluates the vulnerability of a system for any kind of sophisticated attacks. Our approach is based on proposing a **Vulnerability Factor** that accounts for the maximal performance degradation (damage) that sophisticated malicious users can inflict on the system using a specific amount of resources (cost) normalized by the performance degradation attributed to regular users using the same resources.

Vulnerabilities to sophisticated DDoS attacks seem to be an inherent part of existing computer and network systems. Traditionally, these systems have been designed and operated under the underlying fundamental principle that *each user aims at maximizing the performance he/she receives from the system*. While operational strategies have been divided to *social optimization* (optimize operation to the benefit of the overall population) or *individual optimization* (each user acts to improve its performance), see e.g. [8], the basic paradigms were still based on this principle, namely that each user aims at maximizing its own performance. This paradigm does not hold any more in the new "DDoS environment" where some users do not aim to maximize their own performance, but rather to degrade the performance of other users. Thus, there is a *major gap* between traditional assumptions on user behavior and practical behavior in modern networks. Due to this gap it is both natural and inherent that traditional algorithms, protocols and mechanisms will be quite vulnerable in this environment.

Our second contribution, in Section IV, is evaluating the **Hash data structure**, commonly used in network mechanisms, using our vulnerability metric. The sophisticated attack against Hash algorithm was first introduced in [1]. However, [1] concentrated only on Open Hash, using only experimental results on real-life Hash implementations. In contrast, we provide analytic results using our *Vulnerability Factor* metric

[§] This work was supported in part by REMON consortium.
[*] On leave of absence from Tel-Aviv University.
[1] A zombie is a compromised machine that is controlled by a remote hacker The current estimate is that there are millions of machines today, in private homes and institutes, that are zombied.

on the two Hash types: Open Hash and Closed Hash. An important finding of this simple analysis, which can affect practical designs, is that these two common Hash types (which are known to be equivalent according to many performance aspects) drastically differ from each other in their vulnerability to sophisticated attacks.

Our third contribution, (Section V) deals with the Vulnerability of **Queueing policies**, commonly used in network mechanisms. We demonstrate the issue by considering a very simple example of the First Come First Served (FCFS) queue. We show that though attackers, under this mechanism, "must wait like everyone else" and thus have no control of their place in the queue, they can still cause significant damage by using the same resources (traffic) as regular users. The strategy of the malicious users is simple - just submit *large jobs* from time to time. In fact the vulnerability level (using the proposed metric) of this system can be unbounded.

Our fourth contribution (Section VI) shows that in the case of an attack on either Open or Closed Hash, the regular user still suffers from performance degradation or even a total denial of service, even **after the attack has ended** (note that the degradation is not due to the extra load of the attack but due to its sophistication). As far as we are aware, we are the first to point out *post attack* performance degradation. We demonstrate it by using the analysis of the practical case that combines the Hash table with a queue preceding it.

We conclude the paper by comparing the vulnerability of the open and closed hash (Section VII) and a discussion (Section VIII).

II. Sophisticated Attacks

We define *Sophisticated DDoS attack* as an attack in which the attacker sends traffic aiming to hurt a weak point in the system design in order to conduct denial of service[2]. The sophistication lies in the fact that the attack is tailored to the systems design, aiming to increase the effectiveness of the attack. The motivation of the attacker is to minimize the amount of traffic it sends while achieving the same or even better results. Using sophisticated attacks the attacker reduces the cost of attacks i.e., reduces the number of required zombies and reduces the sophistication in coordinating the attack. Moreover, the use of sophisticated attacks increases the likelihood that the attack will succeed in going unnoticed (going Under the Radar) by the DDoS mitigation mechanisms, which are usually based on detecting the fact that the victim was subject to higher-than-normal traffic volume.

Due to the complexity of the applications, they are usually more vulnerable to Sophisticated DDoS attacks. However, the Sophisticated attacks are not just against applications and may also be conducted against protocols (see, for example, attacks against TCP [5]).

Roughly speaking, we can classify the methods of launching sophisticated attacks into three groups, based on the weak design point exploited by the attacker: Worst-Case Exploit,Traffic Pattern Exploit and Protocol Deviation exploit.

1) **Worst-Case Exploit or Complexity attack [1], [9]** - Attacker exploits the worst-case performance of the system which differs from the average case that the system was designed for. Complexity attack problems were described in many different algorithms, such as Hash [1], quicksort [10] regular expression matcher [11], intrusion detection systems [12], [7] and the linux route-table cache [13].

2) **Traffic Pattern Exploit** - Attacker exploits the stochastic worst case traffic pattern to the system. This case is similar to the first one with the difference that the worst case scenario involves a specific traffic arrival pattern of requests from multiple users. This type of attack is demonstrated in the reduction of quality attack papers [3], [4], [7] the Shrew Attack [5] and the stochastic worst case attack on WFQ [2].

3) **Protocol Deviation Exploit** - Attacker forms his own protocol rules, exploiting the fact the protocol was designed using the assumption that all the users obey the rules of the protocol. We are not aware of previous papers that deal with this method of attack, however we have started investigating this method, using the example of Multiple Access protocol (This is part of our future work, see our full paper [14]).

III. The Vulnerability Factor

Our work is based on the observation that the traditional performance analysis of computer systems and algorithms is not adequate to analyze the vulnerability of a system to sophisticated DDoS attack and a new metric is needed.

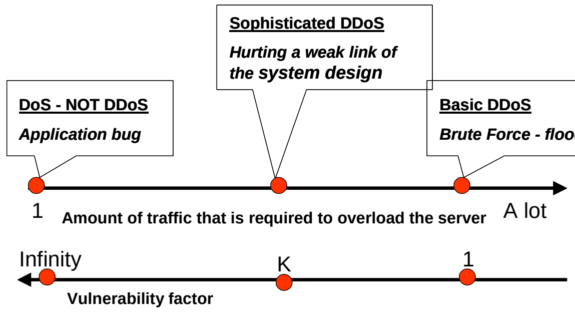


Fig. 1. Vulnerability Factor and Sophisticated DDoS attack

Our proposal for the definition of the Vulnerability Factor of a system is to account for the maximal performance degradation (damage) that sophisticated malicious users can inflict on the system using a specific amount of resources (cost) normalized by the performance degradation attributed to regular users using the same resources. Figure 1 demonstrates the concept of the Vulnerability Factor. Mechanisms that are vulnerable only to brute force attacks will have a Vulnerability Factor of one, while mechanisms that are vulnerable to sophisticated attacks will receive a factor of $K > 1$ indicating that the maximal attacker's power is equivalent to that of K regular users.[3]

[2]or just to significantly degrade the performance (such as Reduction of Quality attacks like [3], [4]).

[3]Placing the DoS attack on the same baseline, the DoS attack (a bug of an application that causes total failure) receives the factor of infinity.

Formally, we define the Vulnerability Factor as follows: Let S be a system in its regular status experiencing usual traffic (load). Let the $usersType$ parameter be equal to either regular users (R) or malicious attackers (M_{st}) with strategy st. Note that we use the plural terms since some of the attack types occur only in specific scenarios of multiple coordinated access of users to the system [3]. Let $budget$ be the amount of resources that the users of $usersType$ spend on producing the additional traffic to the system, i.e., the cost of producing the attack is limited by the $budget$ parameter. The $budget$ can be measured differently in the various models, e.g. as the required bandwidth, or as the number of required computers or as the required amount of CPU and so on. Let $\Delta Perf_S(usersType, budget)$ be the change in the performance of system S due to being subject to additional traffic added to the regular traffic of the system, where the traffic is generated by users from type $usersType$ with resource $budget$. The performance can be measured by different means such as the CPU consumption, the delay experienced by a regular user, as the number of users the system can handle, and so on.

We define the *Effectiveness* of a strategy st on system S as

$$E_{st,S}(budget = b) = \frac{\Delta Perf_S(M_{st}, b)}{\Delta Perf_S(R, b)}, \qquad (1)$$

and the *Vulnerability Factor* V of a system S as:

$$V_S(budget = b) = max_{st} E_{st,S}(b). \qquad (2)$$

A strategy st is said to be an *Optimal Malicious Users Strategy* for system S if the effectiveness of strategy st on S is equal to the *Vulnerability Factor* of S. Note that there can be more than one *Optimal Malicious Users Strategy* and that an *Optimal Malicious Users Strategy* maximizes $\Delta Perf_S(M_{st}, b)$.

The first step for measuring the vulnerability of a system was done in [3] which concentrated only on measuring reduction-of-quality attacks. That paper proposes the measure of *attack potency* which evaluates the amount of performance degradation inflicted by an attacker with specific budget. The reader can observe that this measure is similar to our definition $\Delta Perf_S(M_{st}, budget = b)$. However, the difference between the Effectiveness and the Potency measures is that we normalize the performance degradation the system suffers due to an attacker by the performance degradation the system suffers due to a regular user. Clearly, the potency parameter can be generalized to measure any sophisticated attack, however we think that the *Effectiveness measure* is preferable since it allows to derive meaningful information based on this (dimensionless) number alone[4]. Note that both the Potency and the Effectiveness focus on the power of a specific attack. In contrast, we propose that a more interesting metric is the *Vulnerability Factor* that focuses on the *system* and provides bounds on its performance under any type of attack.

[4]Consider the case of a system with Vulnerability Factor of K. In this case we can deduce that the system can cope with a number of sophisticated attackers, which is only $\frac{1}{K}$ of the number of regular users the system can handle. In contrast, if the potency level is P it means that the attacker obtains P units of damage per one unit of cost; this number is not meaningful without comparing it to the potency of other scenarios. Moreover, the unit of measure (damage per cost) is not fully defined

IV. Sophisticated Attack on Hash Tables

Many network applications, such as the Squid web proxy and the Bro intrusion detection system, use the common Hash data structure. This data structure supports the dictionary operations of Insert, Search and Delete of elements according to their keys. The main idea of a Hash is to transform the key k of an element, using a Hash function h, into an integer that is used as an index to locate a bucket (corresponding to the element) in the array. In the event that two or more keys are hashed to the same bucket, a collision occurs and there are two Hash strategies to handle the collision: Open Hash and Closed Hash. In an Open Hash each bucket in the array points to a linked list of the records that were hashed to that bucket. In a Closed Hash all elements are stored in the array itself; In the insert operation the array is repeatedly probed until an empty bucket is found (for detailed explanations about Open and Closed Hash see [15]).

Consider a Hash system consisting of M buckets and N elements in the Hash, where the Hash function is known to the attackers. In this model the resource-budget of the users is measured by the number of Hash operations (k) they can perform, where Hash operations can be either insert, delete or search.

In our analysis we will use three metrics: i) The number of memory references the k operations require during attack time (*In-Attack Resource Consumption*) and ii) The complexity of performing an arbitrary Hash operation *after the attack has ended*, i.e, after these k Hash-operations are completed (*Post-Attack Operation Complexity*). Note that both metrics, (i) and (ii), are measured by the number of memory references iii) the waiting time of a regular user while preforming an arbitrary Hash Operation *after the attack has ended*, i.e., after the k Hash-operations are completed (*Post-Attack Waiting Time*). The analysis of the third metric appears in Section VI.

In order to carry out the analysis one needs to decide on what type of operations to base the computation of the denominator of Equation 1, that is $\Delta Perf_S(R, b)$. In analyzing the Vulnerability Factor we take the conservative approach and assume that the regular users effect the system by conducting the most time consuming operation types. In both Closed and Open Hash this is the insert operation. Thus we receive a lower bound on the actual Vulnerability Factor.

Let the **Insert Strategy** (INS) denote the malicious users' strategy of performing k insert operations of new elements where all the insert elements hash into the same bucket. We show in the full paper [14] that in Open Hash and Closed Hash systems, the *Insert Strategy* is an *Optimal Malicious Users Strategy* under the *In-Attack Resource Consumption* metric and *Post-Attack Operation Complexity* metric.

A. Open Hash

In this subsection we analyze the vulnerability of the Open Hash table (denoted by OH). We define the load factor[5] L of the Open Hash to be equal to $\frac{N}{M}$, which is the average number of elements in a bucket. We assume each bucket is maintained as a linked list. The insert operation of a new element, which is the most time consuming operation in Hash, requires an expected number of memory references of $L+1$ [15]. The use of the *Insert Strategy* of the malicious users on the Open Hash creates a growing list of elements in one bucket. Thus the malicious users' strategy achieves an $O(k)$ complexity per insert operation instead of the average case of $O(1)$.

In-Attack Resource Consumption Metric

Theorem 1: Under the *In-Attack Resource Consumption* metric the *Vulnerability Factor* of the Open Hash is $V_{OH}(k) = \frac{k+2L+1}{\frac{k+1}{M}+2L+2}$.

Proof: We prove that $\Delta Perf_{OH}(R,k) = k(1+L+\frac{k+1}{2M})$. After the $i-1$th insertion of regular users the load of the table will be $L+\frac{i-1}{M}$, therefore the i-th insertion is expected to require $1+L+\frac{i-1}{M}$ memory references, resulting in total of $k(1+L+\frac{k+1}{2M})$ memory references over k insertions.

Next we prove that $\Delta Perf_{OH}(M_{INS},k) = kL+\frac{k(k+1)}{2}$. The expected memory references which is required by the i-th insertion of malicious users is $L+i$, summing up to $kL+\frac{k(k+1)}{2}$ memory references for the entire insertions sequence. ∎

Post-Attack Operation Complexity Metric

Theorem 2: Under the *Post-Attack Operation Complexity* metric $V_{OH}(k)=1$.

Proof: The proof follows from the fact that the expected length of an arbitrarily linked list in the Hash, after the insertion of the k elements, is always $L+k/M$. This is regardless of their distribution in the buckets. ∎

B. Closed Hash

In this subsection we analyze the vulnerability of the Closed Hash table (denoted by CH). We define the load factor α as the percentage of occupied buckets, i.e. $\frac{N}{M}$. As in Open Hash, in a Closed Hash the most time consuming operation is the insertion of a new element. Each such insertion requires on average approximately $\frac{1}{1-\alpha}$ elementary operations where α is the current load in the Hash (see [15]).

The Vulnerability Factor of both metrics in Closed Hash is influenced by the clustering effect (which also complicates the analysis). In Closed Hash long runs of occupied buckets build up [15]. Due to space limitation we omit the results and proofs, which are given in the full paper [14]. We note that, unlike the Open Hash case, the Vulnerability Factor will be greater than 1 under the **Post-Attack Operation Complexity** metric, since in Closed Hash the cluster that was created due to the attack, has a negative effect on the complexity of a user operation even after the attack has ended. Every insert of an element which is mapped into a bucket in the Cluster (not just specifically the bucket the attackers used) will require to search from its table position to the end of the cluster.

[5]This is the load factor as defined in the literature for data structures [15]; it is different from the traffic load.

V. Sophisticated Attack on Queuing Mechanisms

Queueing mechanisms are major mechanisms used in a variety of computer and communications systems. Their major objective is to control the system operation when it is under heavy *traffic load*, in order to provide good and fair performance to the various users. For this reason it is natural that they become the target of attacks and their vulnerability should be examined.

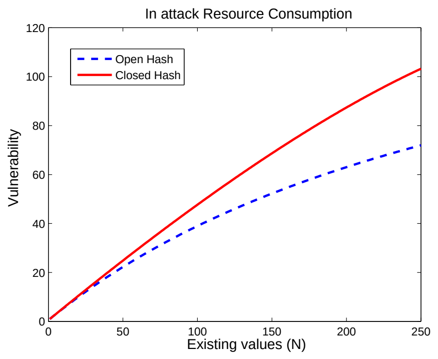
To demonstrate the vulnerability of the queueing mechanism to attacks, if not designed properly, we consider a simple case study consisting of a single server (with single queue) system that operates under the First-Come-First-Served (FCFS) scheduling. This analysis of FCFS is also an useful step for us in order to later evaluate the vulnerability of practical applications that use Hash (combined with a queue) in the Internet (see Section VI). Consider the M/G/1 model with arrival rate λ and service times distributed as random variable X with first two moments $x^{(1)}$ and $x^{(2)}$ and the system utilization is $\rho = \lambda x^{(1)}$. The performance of the system can be evaluated via the mean waiting time experienced by the jobs in equilibrium, which is given by $\lambda x^{(2)}/(2(1-\rho))$.

One way to attack a FCFS queue is to send large jobs. Consider attackers who behave like regular users who pose additional arrival rate λ_1 and whose job size is also distributed like X. Now Consider attackers A who send jobs of size $x = Kx^{(1)}$ at rate $\lambda_2 = \lambda_1/K$, that is, the additional traffic load they pose is identical to the additional load of the regular users, and thus the attack cost from this perspective is the same (identical budget). We prove (in the full paper [14]) that in this case if K is chosen to be large enough, $K(x^{(1)})^2 >> x^{(2)}$, then the Vulnerability Factor can be made as large as one wants, if the system allows arbitrarily large jobs. Thus, if job sizes are not limited the simple FCFS mechanism is highly vulnerable. We therefore may conclude that queueing mechanisms can be quite sensitive to attacks, if not designed properly. Accounting for the Vulnerability Factor of these policies is important in designing them properly

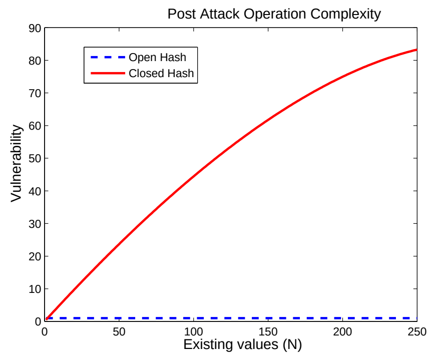
VI. Combining Hash With Queueing

In practical network systems, one would be interested in the combined effect of a hash table and a queue. This is the case where the requests to the hash get queued up in a queue and then are served by the hash table.

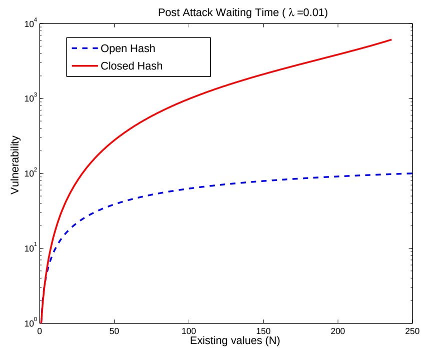
To this end, the queueing analysis given above reveals that the mean waiting time of a request is affected not only by the mean processing time of the other requests but also by their *second moment*. Under the *Post-Attack Waiting Time* metric one should revisit the analysis of the open hash using the *Post-Attack Operation Complexity* metric (Theorem 2). Under that analysis the Vulnerability Factor of Open Hash was determined to be 1 since the *mean* length of the hash linked lists are the same regardless of in which list the items are added.



(a)



(b)



(c)

Fig. 2. Vulnerability comparison of Open and Closed Hash as a function of N, the number of existing elements and where k equals N in Metric (a) *In-Attack Resource Consumption* (b) *Post-Attack Operation Complexity* (c) *Post-Attack Waiting Time* (Y-axis scale is logarithmic)

Nonetheless, since in the *Insert Strategy* , the malicious users force all items into the same list they can increase, *drastically*, the *second moment* of the length of an arbitrary list. As a result, we show that they increase the waiting time of a regular user operation even in Open Hash. For further details see [14].

VII. Open and Closed Hash: Vulnerability Comparison

We use the results derived above to compare the vulnerability of the Open and Closed Hash models. To conduct the comparison we will adopt the common approach that an Open Hash table with M buckets is performance-wise equivalent to a Closed Hash table with $2M$ buckets[6]. Unless otherwise stated, we use the parameters $M = 500$ and $k = N$, meaning that the additional user always doubles the number of the existing values ($k = N$) in the Hash.

Figures 2(a) and 2(b) and 2(c) depict the vulnerability of the Open and Closed Hash as a function of the number of existing elements (N) under the different metrics. The figures demonstrate a striking and very significant difference between the Open and the Closed Hash: The Closed Hash is much more vulnerable to attacks than the Open Hash.

The performance of the open and closed Hash for the regular users *waiting time* metric is provided in Figure 2(c). The figure reveals two interesting and important properties of the system. First, under this metric the Open Hash is significantly more vulnerable then under the *Post-Attack Operation Complexity* metric. This is due to the fact that the delay is proportional to the *second moment* of chain length (and not to the first moment as in Figure (b)). Second, the Closed Hash is drastically more vulnerable (for $N > 237$ the queue overflows) resulting from the compounded effects of: i) the tendency of requests to address large clusters, and ii) the dependency of the mean queueing delay on the *second moment* of the hash operation times. Hence this performance degradation may reach the level of a total denial of service.

[6]It is common to consider them equivalent with respect to the space they require due to the pointers required by the Open Hash.

VIII. Concluding Remarks and Future work

The performance of today's networks and computers is highly affected by DDoS attacks launched by malicious users. This work originated in the recognition that in order to properly evaluate the performance and qualities of these system one must bridge between the world of security and that of quality of service. We addressed this by adding to the traditional performance measures a metric that accounts for the resilience of the system to malicious users.

References

[1] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks," in *Usenix Security*, 2003.
[2] A. Bremler-Barr, H. Levy, and N. Halachmi, "Aggressiveness protective fair queueing for bursty applications (short paper)," in *IWQoS*, 2006.
[3] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the transients of adaptation for RoQ attacks on internet resources," in *ICNP*, 2004.
[4] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, "Reduction of quality (roq) attacks on internet end-systems," in *INFOCOM*, 2005.
[5] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks (the shrew vs. the mice and elephants)," in *ACM SIGCOMM*, 2003.
[6] I. M. M. Guirguism, A. Bestavros and Z. Y, "Reduction of quality (roq) attacks on dynamic load balancers: Vulnerability assessment and design tradeoffs," in *Infocom*, 2007.
[7] R. Smith, C. Estan, and S. Jha, "Backtracking algorithmic complexity attacks against a nids," in *Annual Computer Security Applications Conference December*, 2006.
[8] S. Lippman and S. Stidham, "Individual versus social optimization in exponential congestion systems," *Operations Research*, vol. 25, pp. 233–247, 1997.
[9] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks." http://www.cs.rice.edu/ scrosby/hash/.
[10] M. D. McIlroy, "A killer adversary for quicksort," *Software–Practice and Experience*, pp. 341–344, 1999.
[11] T. Peters, 2003. http://mail.python.org/pipermail/python-dev/2003-May/035916.html.
[12] M. Fisk and G. Varghese, "Fast content-based packet handling for intrusion detectin." UCSD TR CS2001-0670.
[13] F. Weimer, "Algorithmic complexity attacks and the linux networking code." http://www.enyo.de/fw/security/notes/linux-dst-cache-dos.html.
[14] U. Ben-Porat, A. Bremler-Barr, and H. Levy, "Evaluating the vulnerability of network mechanisms to sophisticated ddos attacks," 2007. http://www.faculty.idc.ac.il/bremler/.
[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (Second Edition)*. MIT Press and McGraw-Hill, 2001.